ANL-6576 ANL-6576

Argonne National Laboratory

COMPILER INTO GEORGE ASSEMBLY ROUTINE

by

R. George

ANL-6576
Mathematics and Computers
(TID-4500, 17th Ed.)
AEC Research and
Development Report

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois

COMPILER INTO GEORGE ASSEMBLY ROUTINE

by

R. George

Particle Accelerator Division

June 1962

Operated by The University of Chicago
under
Contract W-31-109-eng-38

COMPILER INTO GEORGE ASSEMBLY ROUTINE

by

R. George
Particle Accelerator Division

ABSTRACT

This program of the GEORGE Assembly Routine (GAR) will accept Fortran-like statements from paper tape and create a GAR language program on tape. This includes the needed calls for common subroutines and the reservations for the named variables and temporaries. The original statements in Fortran are carried along as remarks.

The GAR language program may then be processed in the usual way by the GEORGE Assembly Routine, giving machine-language code.

The level of sophistication of the source language is roughly equal to that of Fortransit or SALT.

INTRODUCTION

Many computer users are accustomed to writing their own programs in Fortran language and then having these programs processed to a particular machine code. The subject of this report is a compilation routine, known as CIGAR, which does one phase of such processing to GEORGE machine code.

This routine will accept Fortran-like statements from a 7-channel punched paper tape and in a single pass create a corresponding GAR language program on paper tape. A package of calls for the most commonly used closed subroutines is always copied into the GAR language program. As the processing proceeds, a list of variable names and temporaries is held in the memory and is afterwards transformed into a list of reservations.

It is desirable to be able to make a comparison between the results of the different processing phases, since this comparison is helpful in debugging the program. To facilitate this, CIGAR will cause the original statements in Fortran language to be carried along into the remarks column of the GAR code that is produced.

There are certain restrictions and conventions which must be observed in programming for this compilation routine. These are discussed at the appropriate places in this report. In addition, the programmer should be cognizant of the restrictions and conventions of GAR, which have been reported elsewhere.

Not all of the abilities of IBM's Fortran are included in CIGAR's order set, but certain other useful orders have been appended. The resultant source language has a level of sophistication roughly equal to that of Fortransit or SALT.

PROGRAM STRUCTURE

Comments

Comments may be written anywhere in the program and are copied exactly into the GAR code as remarks. A comment must begin with a letter, but is otherwise unrestricted as to content, since it does not result in any machine code. A comment is terminated by an <E2>.

Arithmetic Statements

Arithmetic statements must be preceded by a 3-character statement number and 6 spaces <u>or</u> by 9 spaces. Wherever a statement number is used it must be exactly 3 characters, the first of which is a decimal digit. The leading spaces must be present to enable the compiler to differentiate between arithmetic and control statements. Spaces given within a statement are ignored and may be used freely.

Arithmetic statements are each composed of a left-part and a right-part, the 2 being separated by an <=>. The entire statement is terminated by an <E2>.

The left-part may be simple or compound. A simple variable may be defined as either a single variable or a subscripted array. A compound left-part is composed of 2 or more simple variables separated by commas; one should not mix single variable and arrays, nor fixed and floating variables. The left-part must be less than 100 eight-bit characters in length.

The right-part may be almost any legal Fortran combination of numbers, variables, binary operations, and functions. One should not mix fixed and floating quantities arbitrarily since the change of mode is made only on an explicit call for the change-of-mode functions (FIXF, FLOF). For the same reason, right-part mode should agree with left-part mode. The right-part must be less than 250 eight-bit characters in length. It is terminated by an <E2>.

BINARY OPERATIONS. . .

The following symbols are used to denote the basic binary operations in CIGAR. There are no unary operations; a construction such as <=><-> or <(><+> is illegal. The ordering of this list shows the hierarchy which is followed during the formula translation.

- ↑ exponentiation
- * multiplication
- / division
- + addition
- - subtraction

One may write (float↑float) or (float↑fixed) exponentiations. A floating exponent may be a floating-point number, a floating-point variable, or a floating arithmetic expression. The fixed-point exponent in an exponentiation is restricted to a fixed-point number or a fixed-point variable.

The remaining operators may be used either for floating-point arithmetic or for fixed-point arithmetic.

All usages of a particular binary operation symbol at a given level of bracketing are processed to GAR code before proceeding to the next symbol at this level. Scanning for symbols is done from left to right, and from innermost bracket outwards. This means that the sequence

a * b/c * d

would be computed as if it were

(a * b)/(c * d),

but that the sequence

a * b/c/d

would be computed as

((a * b)/c)/d.

FUNCTIONS. . .

When the search for binary operation symbols is exhausted at a given level of bracketing, a test is made for an <F> immediately preceding the left parenthesis. If it is found, this tells us that this is a function. The name which appears to the left of the parenthesis-pair (including terminal <F>) is the name of the subroutine to be called; the computed contents of the parenthesis-pair is used as argument to the function subroutine.

The following functions are built into the running package:

SQRF	Floating point	Square root
ABSF	Floating point	Absolute
SINF	Floating point	Sine - Radians
COSF	Floating point	Cosine - Radians
EXPF	Floating point	Exponential - Base e
LOGF	Floating point	Logarithm - Base e
ATNF	Floating point	Arc Tangent - Radians
FIXF	Floating to Fixed	
FLOF	Fixed to Floating	
SINHF	Floating point	Hyperbolic Sine
COSHF	Floating point	Hyperbolic Cosine
ASINF	Floating point	Arc - Sine
ACOSF	Floating point	Arc - Cosine
RANDF	Floating point	Random number (-1 to +1)
CUBRF	Floating point	Cube root

A programmer may freely use any other name he wishes provided only that the function subroutine that bears this name has somewhere been inserted, presumably with the GAR order (discussed infra).

The mode of the arguments to built-in functions is always floating except for FLOF. The mode of the results of built-in functions is always floating except for FIXF. In any other case, the mode of the results is determined by the first letter of the function name. The mode is floating if this letter lies in the set <A> to <H> or <O> to <Y>. The mode is fixed if the letter lies in the set <I> to <N>.

ARRAYS. . .

Arrayed variables must always be defined by a dimension statement before use. One may do single or double subscripting on either fixed or floating variables. The subscripts must be fixed-point constants or fixed-point variables, or a combination of the 2 in double subscripting. In the evaluation of formulas, the array addresses are computed before the search for binary operation symbols is begun, and therefore one may not put arithmetic expressions as subscripts. There is no 3-dimensional subscripting.

Arrays are stored row by row, forward in memory. An array name must always be accompanied by subscript information, except in READ or PUNCH orders, where it must stand alone. In such orders the entire array is transmitted row by row, moving forward through the memory.

FLOATING NUMBERS...

Within an arithmetic statement, a compiled floating-point number must be at least 3 characters; there must be exactly one digit to the left of the decimal point, and one to 8 digits following it. This may then be followed by the sign of the exponent and a one- or 2-digit exponent. In addition, one must keep floating-point numbers within the range of GAR-assemblable numbers.

The sign preceding the leftmost digit will be considered as a binary arithmetic operation, not as a sign of the number itself.

FIXED NUMBERS...

Compiled fixed-point numbers in arithmetic or control statements may be one to 3 characters in length.

The sign preceding the leftmost digit will be considered as an arithmetic operation, not as a sign of the number itself.

FLOATING VARIABLES...

Floating-variable names may have one to 5 characters, the first of which lies in the set <A> to <H> or <O> to <Y>. One should avoid names starting with <Z>, since such names are created by the CIGAR program. A name starting with <Z> is acceptable either as fixed or floating, but preferably floating if not otherwise determined. Do not use terminal <F> for simple variables.

FIXED VARIABLES...

Fixed-variable names may have one to 5 characters, the first of which lies in the set <I> to <N>. Do not use terminal <F> for simple variables.

Functional Statements

Any arithmetic statement which has simple storage into a single variable of one to 4 characters may be transformed into a functional statement, which defines a closed subroutine, by appending a terminal <F> to the storage-variable name. It also remains a legitimate arithmetic statement.

The statement will be executed and the result stored in the storage variable whenever the right part of any other arithmetic statement includes the name of the functional statement. Control statements may not refer to functional statements.

A functional statement may be placed anywhere in the program, but there is a warning of possible error if there is a call for its use prior to its definition.

Procedure Calls

A procedure call must be preceded by a 3-character statement number and 6 spaces <u>or</u> by 9 spaces. It differs from an arithmetic statement in that no <=> sign appears. It is terminated by an <E2>.

The first 5 characters of the procedure call are taken as the name of the subroutine to be called. This includes internal spaces; therefore, it is often beneficial to check that the first word of the procedure call is 5 or more characters in length.

The subroutine called by a procedure call must somewhere be defined within the program. There is no restriction on the content of such a subroutine, except for assignment of storage variables, if used elsewhere.

Example:

```
    The subroutine:      ......  GAR E2
                         SIGNA   E2 -+- E2 F800 E2/+1 E1
                                 E2 -+- E2 CB01 E2/+1 E1
                                 E2 2AO/E2       E2    E4
The call:                ......  ...SIGNAL FROM TYPEWRITER E2
```

Control Statements

Control statements are preceded by a 3-character statement number and 3 spaces, <u>or</u> by 6 spaces. Leading spaces must be present to enable the compiler to recognize the type of statement. Spaces within a statement are ignored and may be used freely. Control statements are terminated by an <E2>.

The following is a list of control statements allowed in CIGAR.

Absolute GO TO

Computed GO TO

IF

IF SENSE SWITCH

DO

READ

PUNCH

CONTINUE

PAUSE

STOP

DIMENSION

GAR

LOAD

BCD

MODIFY

BEGIN

ABSOLUTE GO TO. . .

This order may refer to any statement number that is defined with the program. Wherever a statement number is used it must be exactly 3 characters, the first of which is a decimal digit.

Examples:

```
GO TO 000
GO TO 532
GO TO 99E
```

COMPUTED GO TO. . .

This order may refer to any one of a number, n, of statement numbers, subject to the control of a fixed-point variable as its index. The value of the index variable may range from 1 to n. If the value of the index variable is 1, control passes to the first statement number; if the value is 2, control passes to the second statement number, etc.

Examples:

```
GO TO (126, 132, 185),  I
GO TO (126, 132, 185)   I
```

IF. . .

Between the parenthesis pair that follows the IF, one may place a single variable name, or <u>any</u> arithmetic expression that does <u>not</u> include further parenthesis.

Either fixed or floating expressions may be used.

Following the parenthesis pair there are 3 statement numbers separated from each other by commas. If the arithmetic expression is negative, control passes to the first; if zero, to the second; if positive, to the third.

If the arithmetic expression is fixed point, the test for zero is made on the leading 12 bits. If the expression is floating point, the test for zero is made on the high-order exponent of the word, resulting in discrimination at about 10^{-20}.

Examples:

```
IF (A) 120, 122, 124
IF (I)  130, 131, 132
IF (A-BIG*THING) 291, 206, 300
IF (J + K * L + I -954) 291, 206, 300
```

IF SENSE SWITCH...

After this order, we place the identifying number of the switch to be used, and a comma. Two statement numbers follow separated by a comma. If the switch is down (on), then control passes to the first address, otherwise, to the second.

Any of the 15 switches is allowed here, but one should remember that A thru F are used to designate input and output devices and modes.*

No parenthesis should be used in this order.

Always set B. P. JUMP when running the object program.

Examples:

```
IF SENSE SWITCH 9, 172, 174
IF SWITCH       A, 097, 099
```

DO...

The DO is immediately followed by a 3-character statement number which we will call the target address. This is immediately followed by a nonarrayed fixed-point variable which is used by the DO loop as an index. This is followed by <=>, followed by lower limit, <,>, upper limit, <,>, and step size.

*When one uses Package AA.

This order may have at its target address a statement that belongs to any one of the following classes:

a = b

READ

PUNCH

CONTINUE

PAUSE

GAR

BCD

The index limits and the index step size may be either a fixed-point decimal number of 1 to 3 digits or a fixed-point variable, not an arithmetic expression.

If the value 1 is to be used for step size, the final comma and the index step size may be omitted.

Fortran rules regarding nesting of DO's must be observed. If one is doing computation with a DO loop, one may not pass control to a subroutine if that subroutine has another DO loop within it, as this would constitute improper nesting.

Examples:

```
DO  128  I  = IIN, IOUT, ISTEP
DO  128  J  = 1,   50,   2
DO  128  K  = 1,   J
```

READ-PUNCH...

Reading and punching are under control of sense switches A B C D E and F at the time of execution of the program.* Switches B and C together determine the physical unit used for input.*

B	C	
0	0	Reader 1
0	1	Magnetic Input
1	0	Reader 2
1	1	Keyboard

*When one uses Package AA.

Switches E and F together determine the physical unit for output.*

E	F	
0	0	Anelex
0	1	Magnetic Output
1	0	Paper Tape
1	1	Typewriter

Switches A and D are used as to determine the maximum number of columns of output information.

A	D	
0	0	8 Columns
0	1	5 Columns
1	0	4 Columns
1	1	Suppress E2's

These settings are tested once at the beginning of execution of the program. If one wishes to make changes during execution one must both change the switches and cause an excursion to the subroutine ZWITC of the running package. The best way is to make a Procedure Call such as:

```
... ... ...ZWITCHES ARE NOW TO BE RESET E2
```

READ 1 orders require the input of fixed-point decimal numbers in the format acceptable to GEO-B-8-202, i.e.,*

Sign number E1

The absolute value of the number must be less than 2000.

READ 2 orders require the input of floating-point decimal numbers in the format acceptable to GEO-B-1-116, i.e.,*

Sign fraction sign of exponent exponent E1

The fractional part may be 1 to 10 digits; the exponent may be 1 to 2 digits. The sign of the number may be <+>, <->, or <sp> and may be preceded by a string of spaces, which in turn may be preceded by a letter string. This is designed so that program output may be read as input provided that the output of <E2> characters was suppressed.

PUNCH 1 orders produce output of fixed-point decimal numbers with absolute value less than 2000. This is output via subroutine GEO-B-6-188.*

*When one uses Package AA.

PUNCH 2 orders produce output of floating-point decimal numbers of the format shown previously. The fractional part of the number will be 9 digits. The exponent will be 2 digits. This output is via subroutine GEO-B-2-117.*

Either single items or arrays may be transmitted by READ-PUNCH orders, but they may not be mixed within a single order. If an array name is given, the entire array is transmitted.

If a PUNCH order is executed for a list of single items (either fixed or floating), the items will be arranged in the maximum number of columns starting at the left of the sheet.*

If a PUNCH order is executed for a list of arrayed variables, the items of each variable will be put in the maximum number of columns, restarting at the left of the sheet for each variable.

Examples:

```
READ 1,    I,  J,  K
PUNCH 1,  ILIST,  JLIST
READ 2,    A,  B,  C
PUNCH 2,  ARRAY,  BLOCK,  CLIQU
```

CONTINUE...

This is a simple No-op.

Example:

```
CONTINUE
```

PAUSE...

At compilation time the last four 8-bit characters which are read following the PAUSE will be transformed into a stored BCD message. At execution time the machine prints the message and stops. Ready, Continuous, Go, causes the machine to proceed in sequence from there.

Examples:

```
PAUSE  0123
PAUSE  DBUG
PAUSE   7777
```

*When one uses Package AA.

STOP. . .

This does the same as PAUSE, except that Ready, Continuous, and Go have no effect.

Examples:

```
STOP    THRU
STOP    DONE
```

DIMENSION. . .

This order serves to define one or more array variables and to reserve space for the elements at the array. At the address that bears the name of the array will be stored a number which describes the array. The B portion of the word holds the total number of reserved words that follow, and the A portion of the word holds the number of columns. If the array is a singly subscripted list, the 2 numbers are equal. This information is referred to every time that an address of an array element is needed; it is used to check whether we have gone beyond the limits of the array. This order is nonexecutable and gives a stop if execution is attempted.

Examples:

```
DIMENSION   ARRAY  (10, 6), BLOCK  (50)
DIMENSION   ILIST  (15), JLIST  (15)
```

GAR. . .

This order permits GAR Language code to be inserted in sequence. The code will be copied exactly down to <E4> which is changed to <E1>.

Examples:

```
GAR   E2
E2   3CO/  E2  CF00  E2  /  E4
```

LOAD. . .

This order permits GAR Language code to be inserted within an array that has previously been defined. Insertion is begun at the first reserved word; one should not permit more words than can be contained in the reserved area. The code will be copied exactly down to <E4>, which is changed to <E1>.

Examples:

```
LOAD ARRAY   E2

E2    E2    DEC    E2    3.781    E1
E2    E2    DEC    E2    4.629    E1
E2    E2    DEC    E2    7.104    E1
E2    E2    DEC    E2    8.615    E4
```

BCD. . .

This order permits any message, less than 250 characters in length, which is intended as output at execution time to be written down on the Fortran coding sheet in the final format. At compilation time, the message is read to <E4>, changed to BCD words, and emitted as GAR code. A call to a message-printing-subroutine is supplied.

At run time, as the control passes in sequence down to this order, it causes execution of the printing routine and a jump around the BCD words themselves.

Examples:

```
        BCD              E2
E2    sp    sp
E2    sp    sp
E2    BENDING MAGNET       1176/PAD
E2    MODIFICATION SIX     31 Nov, 61
E2    sp    sp
E2    sp    sp
E2    ALPHA     BETA     GAMMA     DELTA
E2    sp    E4
```

MODIFY. . .

With this order, one may change arithmetic or control statements in the program at execution time. The following control statements may easily be changed:

Absolute GO TO

Computed GO TO

IF

IF SENSE SWITCH

READ

PUNCH

CONTINUE

PAUSE

STOP

GAR

BCD

The modification of DO's and any order which terminates a DO nest is not recommended.

The order to be modified, the order which follows it in sequence, and the MODIFY order must each have unique statement numbers. The statement number which immediately follows the word MODIFY tells which order is being changed; the second statement number is the number of the one which follows in sequence. The final thing given in a MODIFY is the new statement which is to be inserted during execution time. This may be an arithmetic statement, a procedure call, or a control statement. If it is either of the first two, 3 leading spaces are required.

Examples:

```
200   MODIFY   122,   123,   GO TO 600
201   MODIFY   122,   123,   READ 1, I, J, K, L, M
202   MODIFY   165,   166,   ···ABLE = BAKER
```

BEGIN...

This is always the last compiled statement of a program. It indicates the starting statement number.

Example:

```
BEGIN   100
```

ERROR STOPS

If there is an error in syntax in the Fortran language program, it will probably be detected by CIGAR during the compilation. The machine will print one of the following numbers and the word ERROR on the Anelex, and come to a stop. This list will tell the kind of error that has been detected.

Error	Miscellaneous
1 Error	Line cannot be Classified
2 Error	Variable Name Unsuitable
3 Error	Improper Dimensioning
4 Error	Improper DO Statement
5 Error	Improper IF Statement
6 Error	Variable Name Omitted
7 Error	Statement Number Omitted
8 Error	Improper Modify Order
9 Error	Beyond Limits of List
A Error	Improper Read Order
B Error	Improper Punch Order
C Error	Error in Equation Form
D Error	Too many Characters

SUBROUTINES

The following subroutines as they exist on GAR II as of January 1, 1962, are called and modified by the running package or otherwise introduced into the final program.

NAME	DESCRIPTION	AMD NUMBER
ZAR	Floating Arithmetic	GEO-D-1-115
ZEX	Floating Exponential	GEO-E-7-114
ZIP	Floating Input	GEO-B-1-116
ZOP	Floating Output	GEO-B-2-117
ZQRT	Floating Square Root	GEO-E-8-109
ZICO	Floating Sine-Cosine	GEO-E-11-168
ZLOG	Floating Natural Log	GEO-E-13-170
ZART	Floating Arc-Tangent	GEO-E-15-172
ZIOP	Fixed Integer Output	GEO-B-6-188
ZPII	Fixed Integer Input	GEO-B-8-202
ZROOT	Floating Cube Root	GEO-E-19-207

CHECK LIST

The following is a list of ways in which CIGAR language deviates from IBM 704 Fortran language. The programmer, especially the experienced Fortran programmer, is cautioned to read these restrictions first. I am indebted to John Reynolds for this list.

(1) Leading spaces are used to determine the statement type.

(2) Statement numbers must be exactly 3 characters.

(3) Change of mode occurs only on explicit call for its subroutines.

(4) Not all of Fortran's built-in functions are included.

(5) There is no unary <+> or <->.

(6) The hierarchy of operations does not place multiplication and division on the same level (nor addition and subtraction).

(7) Certain exponentiations are not allowed.

(8) The calls for Fortran's standard functions must be made with the names as listed on page 5.

(9) Arrays are stored in increasing addresses in this way:

$a_{11}, a_{12}, a_{13}, \ldots a_{21}, a_{22} \ldots$

(10) In READ or PUNCH orders, one gives only the name of an array to transmit the entire array.

(11) There is no 3-dimensional subscripting.

(12) Subscripts may not be arithmetic expressions.

(13) Floating-point numbers and fixed-point numbers which are compiled into arithmetic expressions are severely restricted as to format.

(14) Variable names may be one to 5 characters in length, and should not begin with <Z>.

(15) Not all of Fortran's control statements are allowed. (See pages 7 - 8).

(16) The arithmetic expression in an IF statement may not itself contain parentheses.

(17) There is no parenthesis in the IF SENSE SWITCH order.

(18) The limits and step size in a DO order may not be arithmetic expressions.

(19) Nesting rules for DO nests are somewhat more stringent.

(20) There is no FORMAT order. There is one built-in format for fixed-point numbers and one for floating-point numbers.

(21) One may not mix fixed- and floating-variable names, nor single- and arrayed-variable names, within a single READ or PUNCH order.

(22) There is no looping within a READ or PUNCH order.

(23) The final statement of a program must be a BEGIN order.